Julia-based Software-in-the-Loop Offline Co-Simulation for Embedded Control Validation in Power Converters

Nicolas Dimate[1], Sergio A. Dorado-Rojas[2], and Daniel K. Molzahn[2]

***Abstract*—we present a software-in-the-loop co-simulation architecture, implemented within a networked control framework, for validating control software in power converter applications. The proposed architecture combines Julia's symbolic modeling capabilities and high-performance numerical computing with a Python-based controller connected via TCP/IP. This combination provides a modular and flexible environment for software-in-the-loop validation of closed-loop control algorithms. The co-simulation setup is evaluated using a boost converter model regulated by a lead-lag controller. The results show that the developed co-simulation architecture functions correctly and that the proposed validation environment captures implementation-related delays, thereby supporting a more realistic software-based validation of control design before deployment on a real electronic control unit. Furthermore, the potential extension of the proposed architecture to more complex networked control topologies is discussed.**

I. Introduction

The validation of digital control systems associated with power electronic devices has become just as critical as control design itself. This is driven in part by the growing integration of power converters across modern electrified systems, including power systems, electric drives, and transportation, among others. In power systems in particular, the increasing use of converter-interfaced devices is driving the transition toward a more "controls-dominated" paradigm, thereby raising new challenges related to the complexity and uncertainty of system operation, control, and stability [1].

While hardware-in-the-loop (HIL) validation is the ultimate benchmark for reliability, it remains costly and hardware-dependent [2]. Software-based model-oriented validation avoids these constraints at the expense of fidelity: it is prone to model uncertainties and abstracts away implementation-level effects [3]. Validation approaches that bridge this gap, retaining the flexibility of software methods while remaining sensitive to implementation behavior, are therefore needed.

The co-simulation paradigm directly addresses this need, as separating controller execution from plant simulation into independent instances is precisely the structure underpinning software-in-the-loop (SIL) validation. The functional mock-up interface (FMI) standard [4] and HELICS [5] are representative examples of co-simulation frameworks (which could be employed for SIL validation). The former standardizes model exchange interfaces, whereas the latter federates independent simulators through a broker infrastructure for large-scale system studies. While these frameworks provide mature orchestration capabilities across heterogeneous simulation domains, this work focuses specifically on the SIL validation workflow for digitally controlled power converters.

We propose a domain-oriented SIL validation architecture for power electronics, structured around converter modeling, digital controller execution, and validation-oriented analysis. Our main contribution is the definition of a coupling architecture that supports SIL as a valuable intermediate step between early-stage model-based simulation and hardware-based controller validation in power-electronics applications. These validation stages lie within the V-cycle design framework, a methodology that pairs design phases with corresponding validation phases. The proposed SIL co-simulation architecture is positioned within this framework between model- and hardware-based validation.

The open-source implementation integrates a Python-based digital controller with a Julia-based plant model via a TCP/IP lockstep mechanism, complemented by converter-oriented modeling components in Julia that enable the closed-loop evaluation of power electronic devices within the `ModelingToolkit.jl` ecosystem [6]. The implementation leverages Julia for hybrid numeric-symbolic converter simulation, Python for flexible controller implementation, and TCP/IP for simple, transparent inter-process communication.

The remainder of this paper is organized as follows: Section II revisits the V-cycle design framework and positions the various validation methods within it. Section III presents the architecture and describes the main components of the proposed offline SIL co-simulation framework. Section IV introduces the case study used to evaluate the proposed approach, providing details about the system under consideration and the validation context. Section V discusses the results obtained, with a particular emphasis on the relevance of the proposed methodology for validating digital control software and outlining potential future development directions. Finally, Section VI concludes this work.

II. V-Cycle Design and Model-Based Verification

This section revisits the V-cycle design framework and situates the principal validation methods within it, thereby

[1] N. Dimate is an M.Sc. Graduate from ENSTA Paris, Institut Polytechnique de Paris, Paris, France. Email: `junmorenodi@gmail.com`.

[2] S. A. Dorado-Rojas and D. K. Molzahn are with the School of Electrical and Computer Engineering at Georgia Institute of Technology, Atlanta, GA, United States. Emails: {`sadr, molzahn`}`@gatech.edu`.

establishing the conceptual basis for this paper's contribution.

Structure and Stages: As illustrated in Fig. 1, the V-cycle organizes embedded control system development into two sequential phases. In the *project definition* phase, along the descending left arm, the design starts with high-level system requirements, proceeds through system and architectural design, and culminates in detailed module design and code implementation. In the *project validation* phase, along the ascending right-hand side, each design artifact is systematically verified: unit testing validates the implemented modules, integration testing validates the architectural composition, and system and acceptance testing verify compliance with the original system requirements.

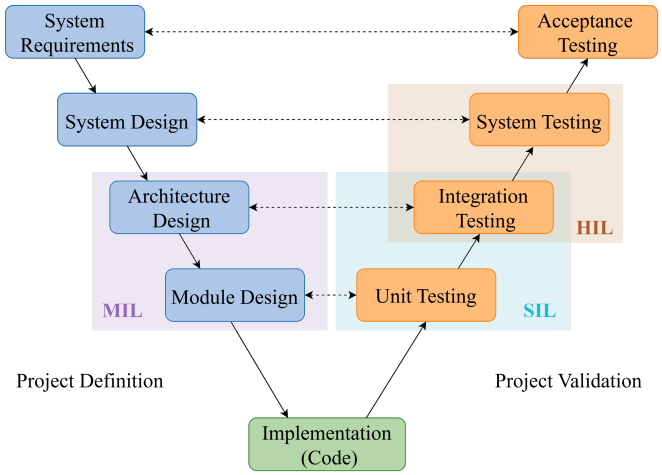


Fig. 1. V-cycle methodology.

Validation Methods: SIL is a software-based implementation-oriented validation method that supports both unit and integration testing by interfacing the final controller implementation with the plant model. Its main objective is to verify that the designed controller has been correctly translated into executable code and that this code is suitable for deployment on the target hardware. The SIL results are then compared with those obtained from the model-in-the-loop (MIL) stage to assess whether the implemented controller preserves the performance and robustness characteristics defined during the model-based design phase. At a later stage, HIL validation addresses integration and system-level testing by executing the controller on an electronic control unit (ECU) and coupling it with dedicated real-time hardware that emulates the plant (or the control in a controller HIL validation). As illustrated in Fig. 2, the progressive application of the validation stages leads to a high degree of maturity before the design is fully integrated into the real system.

SIL occupies an intermediate position between the flexibility of MIL and the realism of HIL. Although it remains simulation-based and typically relies on a first-principles plant model, SIL differs from the conventional MIL framework in that the controller is represented not by a high-level functional model, but by its actual software implementation executed in a dedicated environment. As such, it serves as an effective complement to HIL testing during early validation phases, as it does not rely on dedicated hardware infrastructure [3].

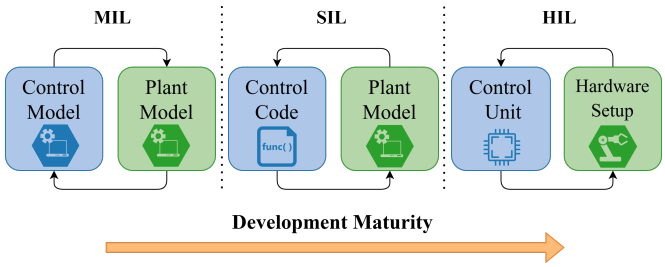


Fig. 2. MIL, SIL, and HIL frameworks: control and system representations.

Offline Versus Real-Time SIL Testing: Two types of SIL implementation are commonly found in the literature, depending on the nature of the testing:

- **Offline testing**: enables functional validation of control algorithms without time constraints. Its primary objective is to verify that the functional behavior defined during the design phase is preserved at the software level.
- **Real-time (RT) testing**: enforces timing requirements to assess execution behavior under realistic operating conditions. The model senses the actual signals but is highly dependent on the communication interface to ensure correct synchronization among all the parts.

This work focuses exclusively on offline SIL, as it enables faster development iterations and is not constrained by RT requirements. This makes it particularly suitable for broad exploration of test scenarios, including faults, edge cases, and variations in model parameters.

Even though SIL is less representative of physical hardware effects than HIL, it provides a strong foundation for system validation when based on reliable dependencies, an accurate plant model, and solid numerical performance [7]. Under these conditions, SIL increases confidence in early-stage design while providing a flexible, scalable, and reusable validation environment. This is particularly relevant in the context of the growing demand for power electronics integration within an increasingly digital and "controls-dominated" network [8]. In this sense, SIL is not only a complementary validation method, but also a practical means of shifting part of the validation effort toward earlier stages of the development process.

From a V-cycle perspective, this upstream shift is valuable because, although following the full methodology significantly improves system reliability, its complete deployment remains both time-consuming and costly. From both industrial and research perspectives, this creates a strong incentive to rely more heavily on software-based validation. In particular, the high cost of HIL platforms and the rapid increase in converter-grid interaction scenarios make exhaustive hardware-based validation cumbersome and time-consuming, thereby limiting both coverage and testing speed [2]. This limitation becomes

particularly important when shorter development cycles and faster design iterations are required.

As a result, strengthening SIL validation enables earlier detection of integration issues, lowers validation cost, and alleviates the hardware-based testing bottleneck.

III. Validation Interface: A Networked Control Approach

The proposed co-simulation architecture is based on the concept of networked control (NC), a paradigm widely adopted in industrial automation, Industry 4.0, and the internet of things (IOT) [9]. In this paradigm, the controller and the plant are interconnected via a communication network that closes the control loop, making the communication interface a third, intermediary component of the overall system. This structure provides a natural path for SIL implementations: by separating controller execution from plant simulation into independent processes, it captures implementation-level effects, including communication and computation delays that model-based methods neglect. A basic NC system is illustrated in Fig. 3.

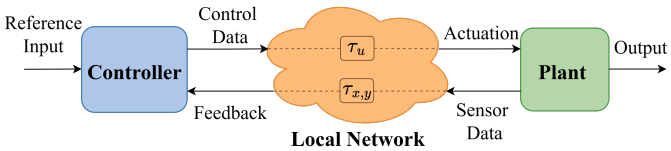


Fig. 3. Networked control architecture.

Based on this framework, the system architecture adopts a client/server model [9], as shown in detail in Fig. 4. The system consists of three main modules, described next.

A. Julia Server: Plant Model Implementation

The plant model in the proposed SIL architecture is implemented in Julia (v1.12.5), which was selected for its combination of high-performance numerical execution with an ecosystem for symbolic modeling and differential equation solving [6], [10]. Julia offers an open-source solution for representing power electronic systems while preserving a "white-box" philosophy, which is not always guaranteed in proprietary software.

As shown in Fig. 4, the Julia module serves as the plant-side server, hosting the system model, receiving the control input $\boldsymbol{u}(t)$ from the Python client, solving the plant dynamics, and returning the corresponding state and output measurements $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$, to the controller for the next control update. The workflow begins with the model instantiation and compilation using the native `ModelingToolkit.jl` functions. The compiled model is then used to formulate an ordinary differential equation (ODE) problem, which is executed within a lockstep simulation loop.

The plant model is solved over successive intervals of duration Δt under a fixed control input $\boldsymbol{u}(t)$. The interval Δt, denoted as the *synchronization step*, regulates data exchange within the lockstep framework. The control input is held constant for $\lfloor T_{ctrl}/\Delta t \rfloor$ integration steps, after which the controller is updated using the resulting plant state. To preserve synchronization, the controller computes the control action at each Δt instant and sends it to the Julia server for use as the plant input during the next integration step. Within each Δt interval, the plant model is solved as a continuous-time (CT) dynamical system using adaptive time stepping with a maximum step size bounded by $dt_{\max}$ to prevent the solver from crossing switching discontinuities.

B. Python Client: Control Module Implementation

The control module within the proposed SIL framework was implemented in Python (v3.13.5). Although it is not constrained by design to a specific programming language, Python was selected as a practical environment for control testing and implementation due to its widespread use,

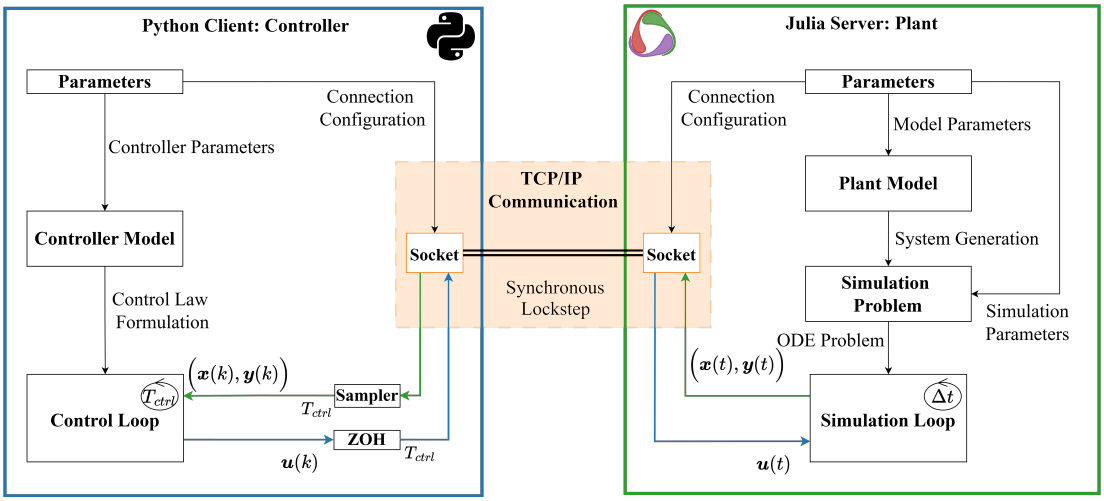


Fig. 4. Proposed SIL co-simulation architecture based on a synchronous, lockstep TCP communication interface between a DT Python controller and a CT Julia-based plant model.

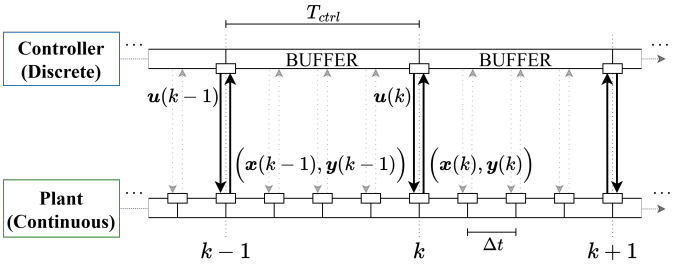


Fig. 5. Timing scheme of the communication layer. Black arrows denote control updates, and gray-dashed arrows denote synchronization exchanges.

compatibility with existing control libraries, and support for rapid prototyping. In the client/server architecture, the Python module serves as the client-side component, hosting the control algorithm, receiving plant outputs from the Julia server, and computing control commands. The workflow consists of two stages: controller definition and control-loop execution. First, the controller is instantiated according to a predefined structure (e.g., PI, PID, or lead-lag) with selected control gains and parameters. During control loop execution, the plant measurements are compared with the reference values to compute control actions, which are then transmitted back to the Julia server.

The controller operates in the discrete-time (DT) domain with sampling period T_{ctrl}. The transmission is coordinated through the *synchronization step* Δt, providing a common timing reference between DT controller and CT plant. Control commands are held constant between samplings using a buffer-based zero-order hold (ZOH) mechanism. Although a control value is transmitted every Δt, the controller output is updated only once per sampling period T_{ctrl}, as illustrated in Fig. 5.

C. TCP/IP Socket Communication

The communication layer enables bidirectional data exchange between the Python controller and the Julia plant server through TCP/IP, a widely supported network protocol. In the present implementation, communication is routed through the local IP address `127.0.0.1` using the *loopback* interface, so no physical network link is present, and all packets are handled internally by the operating system. This setup provides a reliable ordered end-to-end byte-stream transmission between both modules, making it well-suited for an offline SIL framework [11].

Synchronization is enforced at the application level through a synchronous lockstep scheme. At each *synchronization step*, Δt, the controller computes and encodes the control action as a JSON string, then sends it to the plant server. The server receives and decodes the JSON message, advances the model, and returns the updated states and outputs, $(\boldsymbol{x}(t), \boldsymbol{y}(t))$, in the same way, to complete a communication cycle.

Although JSON is generally unsuitable for RT applications because its text-based structure introduces serialization overhead and can become a bottleneck under high message rates, it remains appropriate for the present application, given the simple message structure and relaxed offline timing requirements. Future work may adopt a more suitable data format for RT applications.

The resulting architecture enables closed-loop offline validation in which the controller executes as actual software, communicates with the plant model through a structured interface, and is subject to the timing constraints of a real digital implementation. These conditions are examined next in Section IV, where the co-simulation setup is applied to a boost converter under voltage-mode control (VMC) with a discrete-time lead-lag compensator.

IV. Case Study: Boost Converter

We consider a PV boost converter as the case study, shown in Fig. 6. The converter parameters are as follows: nominal load resistance $R_\ell = 90\,\Omega$, switching frequency $f_s = 50\,\text{kHz}$, input voltage $V_{in} = 370\,\text{V}$, output voltage $V_o = 600\,\text{V}$, $L = 3.5\,\text{mH}$, $R_L = 100\,\text{m}\Omega$, $C = 3.3\,\text{mF}$, and $R_C = 10\,\text{m}\Omega$. The switch Q is assumed to have negligible on-resistance, while the diode D is treated as ideal. The converter operates in continuous conduction mode (CCM). Further details on the considered converter and its control can be found in [12], where it is examined within a DT modeling framework, particularly relevant in the present context for analyzing communication- and computation-induced delays in NC applications.

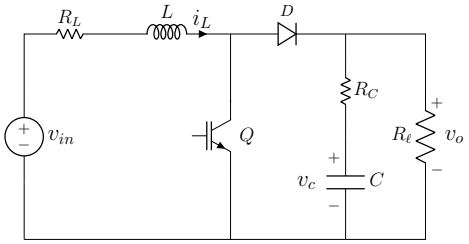


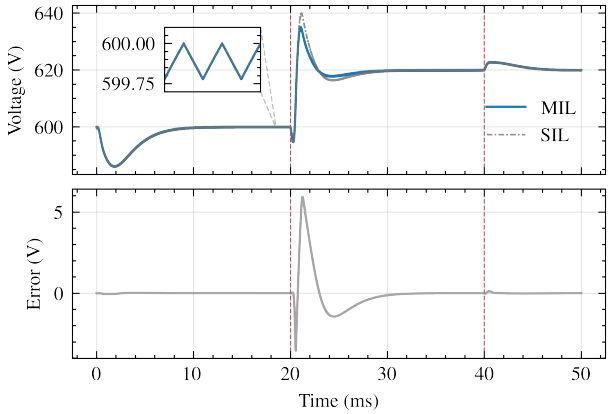
Fig. 6. Circuit diagram of a boost converter.

A VMC scheme is employed for the control of the boost converter. The corresponding controller is a discrete-time lead–lag compensator, $G_{c_{dis}}(z)$, obtained through an algebraic frequency-domain design procedure. The compensator is first synthesized in the continuous-time domain as $G_c(s)$, with a phase margin of PM $= 45°$ and a cutoff frequency $f_c = 2\,\text{kHz}$ in accordance with the guideline $f_c < f_s/20$ [13]. The result is discretized via the bilinear (Tustin) transformation, using the converter's switching period $T_s = 1/f_s$ as sampling period, T_{ctrl}. The implemented digital lead-lag compensator[1] is:

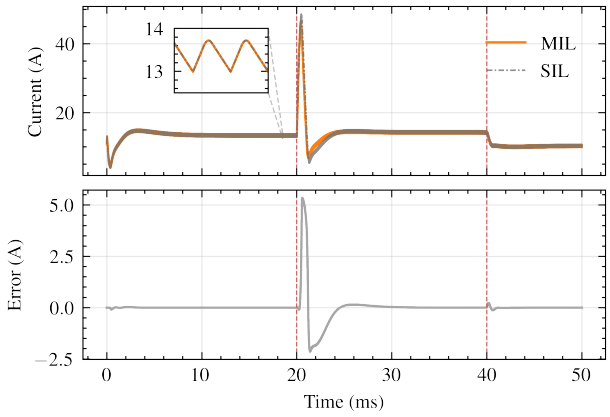
$$G_{c_{dis}}(z) = \frac{0.09114z^2 - 0.1797z + 0.08856}{z^2 - 1.782z + 0.7819}. \qquad (1)$$

The validation objective is to verify that the SIL architecture correctly reproduces the regulated MIL behavior, particularly

[1]Coefficients are rounded here for brevity. The implementation uses full-precision values to preserve accuracy in the discrete-time simulation.



(a) Capacitor voltage $v_C(t)$.



(b) Inductor current $i_L(t)$.

Fig. 7. Closed-loop responses of capacitor voltage and inductor current under a reference step at $t = 20\,\text{ms}$ and a load step at $t = 40\,\text{ms}$. Vertical red lines indicate the event instants.

the output voltage, while capturing implementation-level effects, such as communication-induced delays, in the control action and unregulated converter states (i.e., inductor current).

V. Results and Analysis

The MIL test, implemented entirely in Julia, is adopted as the functional baseline against which the SIL co-simulation is evaluated. Both the MIL and SIL use the DT controller presented in Section IV, with a sampling period T_{ctrl}. The *synchronization step* of the lockstep mechanism is set to $\Delta t = T_s/100$. The converter plant is solved using the adaptive solver `Rodas5P` with a step size bounded above by $dt_{\max} = \Delta t$.

Two experiments are considered: 1) a *reference step* in which the reference voltage for the converter output changes from $V^* = 600\,\text{V}$ to $V^* = 620\,\text{V}$ at $t = 20\,\text{ms}$, and 2) a *load step*, in which the load resistance changes from its nominal value of $R_\ell = 90\,\Omega$ to $R_\ell = 135\,\Omega$ at $t = 40\,\text{ms}$.

The resulting responses of the converter are shown in Fig. 7. The SIL framework preserves the expected steady-state control behavior under disturbance conditions, while observable differences appear during the transient response. The zoomed views confirm steady-state agreement, with low interface-induced error and accurate reproduction of both the voltage and current ripple. Deviations become more visible during rapid changes, where synchronization and software-execution effects are expected to be more influential. To quantify these differences, the root-mean-square error (RMSE) is used:

$$\text{RMSE}_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{\text{SIL},i} - x_{\text{MIL},i})^2}. \tag{2}$$

The normalized root-mean-square error (NRMSE) is then calculated by normalizing the RMSE with respect to the signal range, defined as the difference between the maximum and minimum values over each considered interval, ΔT. This normalization enables comparison of the mismatch across operating regions relative to the corresponding signal range:

$$\text{NRMSE}_x = \frac{\text{RMSE}_x}{(x_{\text{MIL,max}} - x_{\text{MIL,min}})}\bigg|_{\Delta T}. \tag{3}$$

The results are reported in Table I for five different time intervals. The intervals from 10 to 15 ms and from 30 to 40 ms were omitted to avoid biasing the analysis by mixing transient and quasi-steady-state behavior. The defined time intervals are:

- ΔT_1: startup from initial condition, from 0 to 10 ms
- ΔT_{SS}: steady-state window, from 15 ms to 20 ms
- ΔT_2: reference-tracking transient, from 20 to 30 ms
- ΔT_3: load-disturbance transient, from 40 ms to 45 ms
- ΔT_{FS}: full simulation interval, from 0 to 50 ms

TABLE I
NRMSE ACROSS EVALUATION INTERVALS

Signal	NRMSE (%)				
	ΔT_1	ΔT_{SS}	ΔT_2	ΔT_3	ΔT_{FS}
Capacitor voltage, v_C	0.00	0.00	0.30	0.01	0.13
Inductor current, i_L	0.12	0.01	9.24	0.52	4.75
Duty cycle, d	1.47	0.02	10.37	0.82	5.00

All values are expressed in %.

The low NRMSE for $v_C(t)$ across all intervals confirms that the primary validation objective is met: the SIL framework reproduces the regulated output with high fidelity relative to the MIL baseline. The larger deviations in $d(t)$ and $i_L(t)$ should not be interpreted as validation failures; rather, they reflect implementation-level effects (e.g., communication-induced delays) that the controller must compensate for to maintain voltage regulation. Because $i_L(t)$ is unregulated, these effects propagate into its dynamics without correction, making it a more sensitive indicator of interface-induced discrepancies.

During ΔT_1, both implementations exhibit a similar behavior, with negligible error, mainly in the control action, and a small transient. The largest discrepancy is observed in ΔT_2, where overshoot and undershoot appear after the reference step. The waveforms suggest a small effective delay in the SIL response, which becomes more noticeable during rapid transients. This effect is visible in Fig. 8, where the SIL

duty-cycle response lags the MIL baseline. The resulting delay in the control action propagates to the unregulated state $i_L(t)$.

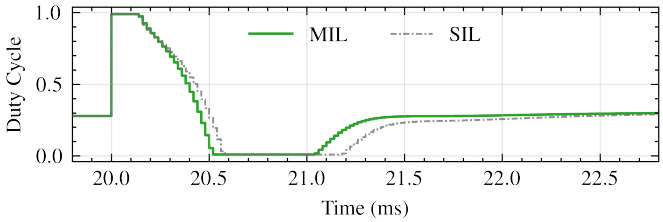


Fig. 8. Duty-cycle response $d(t)$ over the reference-step transient (ΔT_2).

In ΔT_3, the controller rejects the load disturbance effectively, limiting the transient magnitude and thus reducing the difference between MIL and SIL. Under steady-state conditions, in ΔT_{SS}, the SIL response is nearly identical to the MIL baseline, with negligible plant-state error below the reported resolution for $v_C(t)$. Finally, ΔT_{FS} provides a global measure of the agreement over the full simulation horizon.

These tests show that MIL and SIL predict the same qualitative closed-loop response under representative disturbances, while also highlighting the implementation-induced effects captured by the SIL setup. The observed discrepancies may be due to implementation-induced effects, such as numerical representation, DT execution, data conversion, and, especially, the communication-layer delay observed in the duty-cycle response. In the presented case, these effects remain small and do not compromise the regulated output response, but are still measurable in the duty cycle and in the unregulated inductor current during fast transients. The results highlight the value of SIL validation for quantifying implementation-induced discrepancies and for indicating whether they remain acceptable. In applications with tighter performance margins and stricter performance requirements, the interface effects revealed by the SIL simulation could justify an additional design iteration.

VI. Conclusions

This article addressed the gap between model-based control validation and deployment on a real platform by proposing an offline SIL co-simulation framework for power electronics applications. The key architectural choice, separating the controller and the plant into independent, communicating software processes, naturally introduces implementation-level conditions absent from conventional MIL testing while preserving the flexibility of offline simulation.

The case study confirmed that the framework meets its validation objective: the regulated output is reproduced with high fidelity relative to the MIL baseline, while measurable discrepancies in the control action and unregulated states expose the effects of communication-induced delays and software execution. These discrepancies are not incidental artifacts; they are the informational content that SIL validation is designed to produce, enabling the designer to assess implementation effects before committing to hardware.

In this sense, the proposed environment fills a well-defined intermediate role in the development process: it extends the confidence of MIL validation by introducing realistic software execution conditions, without the complexity of HIL testing. In applications with tighter performance margins, the effects quantified here could justify an additional design iteration.

Future Work: The proposed architecture can benefit from virtualizing both the plant and the controller, enabling more complex NC architectures, as supervisory networked and cloud-based control [14]. Such extensions could improve scalability and computational capability for demanding controllers, such as model predictive control (MPC) or convex optimization-based control, and for numerically intensive nonlinear plant simulations [15]. However, their adoption requires careful consideration of latency, packet losses, and their impact on closed-loop performance and RT feasibility.

Data Availability

The repository will be released upon article acceptance.

References

[1] F. Dörfler and D. Groß, "Control of low-inertia power systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 6, pp. 415–445, May 2023.

[2] A. Derviskadic, B. Kurth, and J. Hernandez, "Advancing HVDC control & protection systems: The role of software-in-the-loop approaches," in *AEIT HVDC International Conference (AEIT HVDC)*, 2025.

[3] X. Chen, M. Salem, T. Das, and X. Chen, "Real-time software-in-the-loop simulation for control performance validation," *Simulation*, vol. 84, pp. 457–471, Aug. 2008.

[4] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, "The functional mockup interface for tool-independent exchange of simulation models," in *8th International Modelica Conference*, Mar. 2011, pp. 105–114.

[5] T. D. Hardy, B. Palmintier, P. L. Top, D. Krishnamurthy, and J. C. Fuller, "HELICS: A co-simulation framework for scalable multi-domain modeling and analysis," *IEEE Access*, vol. 12, pp. 24 325–24 347, 2024.

[6] C. Rackauckas and Q. Nie, "DifferentialEquations.jl–A performant and feature-rich ecosystem for solving differential equations in Julia," *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[7] C.-M. Lai and Y.-J. Lin, "Novel offline software-in-the-loop simulation technique for modular single-phase flyback current source grid-tie inverter system," *IEEE Access*, vol. 9, pp. 100 814–100 826, 2021.

[8] R. A. Acosta-Rodríguez, F. H. Martinez-Sarmiento, G. A. Múñoz-Hernandez, G. Mino-Aguilar, E. A. Portilla-Flores, P. A. Niño-Suarez, and O. J. Salcedo-Parra, "Validation of sliding mode and passivity control in high-power quadratic buck converter through rapid prototyping," *IEEE Access*, vol. 12, pp. 8668–8699, 2024.

[9] W. jong Kim, K. Ji, and A. Srivastava, "Network-based control with real-time prediction of delayed/lost sensor data," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 1, pp. 182–185, 2006.

[10] Y. Ma, S. Gowda, R. Anantharaman, C. Laughman, V. Shah, and C. Rackauckas, "ModelingToolkit: A composable graph transformation system for equation-based modeling," *arXiv:2103.05244*, 2021.

[11] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. USA: Prentice Hall Press, 2010.

[12] N. Dimate, S. A. Dorado-Rojas, and D. K. Molzahn, "On the impact of PWM modulation strategies on the discrete-time modeling of a boost converter," in *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, Oct. 2025.

[13] R. W. Erickson and D. Maksimović, *Fundamentals of Power Electronics*, 3rd ed. Springer, 2020.

[14] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Control-over-the-cloud: A performance study for cloud-native, critical control systems," in *13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 57–66.

[15] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.